Internet Programming in Python - Week 8

Large web frameworks - Django

http://en.wikipedia.org/wiki/File:Django_Reinhardt_(Gottlieb_07301).jpg

http://djangopony.com/

Brian Dorsey
brian@dorseys.org

Guest Speaker

(30)

Ted Leung

http://www.sauria.com/blog/

API Talks

(20)

talks

Questions and review

(20)

some thoughts from the assignments

turn in here: `http://is.gd/uwipip_week7`

Assignment

- build a small website for something you (or your friends/family) collect or love
- admin UI is enough for editing
- list of all items
- item detail pages
- pivot on at least one item detail (ex: for books, click author, get list of books by that author)
- stretch: configure the admin UI to make it really clean
- stretch: searching
- stretch: items are owned, and can be loaned to other people.
- stretch: make it look decent - use design templates if needed

pick at least one

Break

(10)

ANNOUNCEMENTS

The 10 Python Conferences Happening at PyCon 2011

Virtual Track #1: Django

http://pycon.blogspot.com/2011/02/10-python-conferences-happening-at.html

A moment to reflect

next week

- project layout
  - apps vs projects
  - on disk organization
- more admin customization
- django.contrib
  - optional, de facto standard implementations of common patterns
- urls
  - named urls
  - reverse(), permalink, {% url %}
- database API
  - related objects
  - QuerySets
  - raw SQL
  - non-relational DBs
- templates
  - overview
  - template inheritance
  - custom template tags
- caching
- forms
  - overview
  - API
  - fields and widgets
  - model based forms
- signals
- unicode & encodings
  - defaults
  - how to change it
- middleware
- good stuff from django-admin and manage
- a taste of testing
  - unit test
  - Django extras
  - fixtures
- syndication? (rss, etc)
- extra
  - geodjango
  - stand alone scripts which use the database API
- other people's apps
  - where to find them
  - which ones should we look at first?
- deploying django
  - webfaction setup demo
- Django 1.3
  - what's new
  - what's being removed

Lecture A
(25)

Django 1.3

WordPress.org

Home Showcase Extend About Docs Blog

WordPress News

"Django"

WordPress 3.1, lots of fun

Posted February 22, 2011 by Matt Mullenweg. Filed under Releases.

The long-awaited fourteenth release of WordPress is now available. WordPress 3.1 "Django" is named in honor of the jazz guitarist Django Reinhardt. Version 3.1 is available for download, or you can update from within your dashboard.

This release features a lightning fast **redesigned linking workflow** which makes it easy to link to your existing posts and pages, an **admin bar** so you're never more than a click away from your most-used dashboard pages, a **streamlined writing interface** that hides many of the seldom-used panels by default to create a simpler and less intimidating writing experience for new bloggers (visit Screen Options in the top right to get old panels back), and a **refreshed blue admin scheme** available for selection under your personal options.

There's a bucket of candy for developers as well, including our new **Post Formats support** which makes it easy for themes to create portable tumblelogs with different styling for different types of posts, **new CMS capabilities** like archive pages for custom content types, a **new Network Admin**, an overhaul of the import and export system, and the ability to perform **advanced taxonomy and custom fields queries**.

With the 3.1 release, **WordPress is more of a CMS than ever before**. The only limit to what you can build is your imagination.

http://wordpress.org/news/2011/02/threeone/

Y **Hacker News** new | threads | comments | ask | jobs | submit

▲ WordPress names their 3.1 release: Reinhardt (wordpress.org)

160 points by yahelc 5 days ago | 106 comments

▲ 146 points by photomatt 5 days ago | link

Hey guys -- for the past 7 years we've named every release after a jazz musician, as you can see here:

http://wordpress.org/about/roadmap/

Usually no one notices, and the codename isn't even in the tweets or announcement email, just a sentence that clearly links to the jazz musician's Wikipedia page. (It's not like Ubuntu codenames that are widely used.) We also put a fun plug for djangoproject.org to the end of the "future" section, which hopefully will also introduce some new people to their project. I haven't seen anyone tweeting the codename except to complain, so I don't think this is going to confuse anybody think we're rewriting in Python. (However much they may want that.)

That said, apologies for the unintended controversy. In hindsight, we probably should have used "Reinhardt" to have the same effect of honoring one of our favorite musicians without anyone getting confused with a fellow Open Source project.

Now to work on 3.2 Sinatra! (Kidding, Ruby folks.)

reply

▲ 140 points by jacobian 5 days ago | link

Hi Matt --

I know this really isn't a big deal, and I really don't want to be a dick about it, but in this age of Google I would really appreciate it if you'd change the name. I think if we codenamed Django 1.3 "Wordpress" you'd feel similarly, right? I really don't want this to turn into A Thing, so how about you just see this as a small favor for a fellow open source hacker and I owe you beer/whiskey/whatever?

Please feel free to get in touch personally if you want to talk further - I'm jacob@jacobian.org.

(I'm Jacob Kaplan-Moss, one of the lead Django devs.)

reply

http://news.ycombinator.com/item?id=2254299

Founder Daily

ARCHIVES RSS FEED

Django names their 1.3 release: 'WordPress'

In Technology on February 23, 2011 at 11:08 am

Update: WordPress has yielded and renamed their release Reinhardt, the last name of jazz musician Django Reinhardt. In response, Django has changed their release name to Automattic, the last name of WordPress.

Django announced the name of their long-awaited 1.3 update: *WordPress*.

The move comes on the same day, shockingly, as the niche software organization WordPress named their 3.1 release *Django*.

"You would think it could be confusing," said Marie Adamski, a Django developer and contributor to the project. "Django 1.3 is called WordPress and WordPress 3.1 is called Django. But it's not."

http://founderdaily.com/2011/02/23/django-names-their-1-3-release-wordpress/

NEWS

WP

WordPress 3.1 "Django" becomes WordPress 3.1 "Reinhardt"

8

By Ryan Imel *5 days ago*

WordPress.org

Home Showcase Extend About Docs Blog

WordPress News

WordPress 3.1, lots of fun

Posted February 22, 2011 by Matt Mullenweg. Filed under Releases.

The long-awaited fourteenth release of WordPress is now available. WordPress 3.1 "Reinhardt" is named in honor of the jazz guitarist Django Reinhardt. Version 3.1 is available for download, or you can update from within your dashboard.

This release features a lightning fast **redesigned linking workflow** which makes it easy to link to your existing posts and pages, an **admin bar** so you're never more than a click away from your most-used dashboard pages, a **streamlined writing interface** that hides many of the seldom-used panels by default to create a simpler and less intimidating writing experience for new bloggers (visit Screen Options in the top right to get old panels back), and a **refreshed blue admin scheme** available for selection under your personal options.

There's a bucket of candy for developers as well, including our new **Post Formats support** which makes it easy for themes to create portable tumblelogs with different styling for different types of posts, **new CMS capabilities** like archive pages for custom content types, a **new Network Admin**, an overhaul of the import and export system, and the ability to perform **advanced taxonomy and custom fields queries**.

With the 3.1 release, **WordPress is more of a CMS than ever before**. The only limit to what you can build is your imagination.

http://wordpress.org/news/2011/02/threeone/

what's new?

(highlights only)

http://docs.djangoproject.com/en/dev/releases/1.3/

planned for release
sometime this week

http://www.djangoproject.com/weblog/2011/feb/16/django-1_3-release-schedule-update-4/

Logging

Django 1.3 adds framework-level support for Python's logging module. This means you can now easily configure and control logging as part of your Django project. A number of logging handlers and logging calls have been added to Django's own code as well – most notably, the error emails sent on a HTTP 500 server error are now handled as a logging activity. See the documentation on Django's logging interface for more details.

Extended static files handling

Django 1.3 ships with a new contrib app `'django.contrib.staticfiles'` to help developers handle the static media files (images, CSS, Javascript, etc.) that are needed to render a complete web page.

unittest2 support

Python 2.7 introduced some major changes to the unittest library, adding some extremely useful features. To ensure that every Django project can benefit from these new features, Django ships with a copy of unittest2, a copy of the Python 2.7 unittest library, backported for Python 2.4 compatibility.

To access this library, Django provides the `django.utils.unittest` module alias. If you are using Python 2.7, or you have installed unittest2 locally, Django will map the alias to the installed version of the unittest library. Otherwise, Django will use it's own bundled version of unittest2.

To use this alias, simply use:

```
from django.utils import unittest
```

wherever you would have historically used:

```
import unittest
```

If you want to continue to use the base unittest libary, you can -- you just won't get any of the nice new unittest2 features.

CSRF exception for AJAX requests

Django includes a CSRF-protection mechanism, which makes use of a token inserted into outgoing forms. Middleware then checks for the token's presence on form submission, and validates it.

Prior to Django 1.2.5, our CSRF protection made an exception for AJAX requests, on the following basis:

To remedy this, Django will now apply full CSRF validation to all requests, regardless of apparent AJAX origin. This is technically backwards-incompatible, but the security risks have been judged to outweigh the compatibility concerns in this case.

Additionally, Django will now accept the CSRF token in the custom HTTP header X-CSRFTOKEN, as well as in the form submission itself, for ease of use with popular JavaScript toolkits which allow insertion of custom headers into all AJAX requests.

No more naughty words

Django has historically provided (and enforced) a list of profanities. The comments app has enforced this list of profanities, preventing people from submitting comments that contained one of those profanities.

Unfortunately, the technique used to implement this profanities list was woefully naive, and prone to the Scunthorpe problem. Fixing the built in filter to fix this problem would require significant effort, and since natural language processing isn't the normal domain of a web framework, we have "fixed" the problem by making the list of prohibited words an empty list.

If you want to restore the old behavior, simply put a `PROFANITIES_LIST` setting in your settings file that includes the words that you want to prohibit (see the commit that implemented this change if you want to see the list of words that was historically prohibited). However, if avoiding profanities is important to you, you would be well advised to seek out a better, less naive approach to the problem.

deprecated?

http://docs.djangoproject.com/en/dev/releases/1.3/

Test client response `template` attribute

Django's test client returns `Response` objects annotated with extra testing information. In Django versions prior to 1.3, this included a `template` attribute containing information about templates rendered in generating the response: either None, a single `Template` object, or a list of `Template` objects. This inconsistency in return values (sometimes a list, sometimes not) made the attribute difficult to work with.

In Django 1.3 the `template` attribute is deprecated in favor of a new `templates` attribute, which is always a list, even if it has only a single element or no elements.

`mod_python` support

The `mod_python` library has not had a release since 2007 or a commit since 2008. The Apache Foundation board voted to remove `mod_python` from the set of active projects in its version control repositories, and its lead developer has shifted all of his efforts toward the lighter, slimmer, more stable, and more flexible `mod_wsgi` backend.

If you are currently using the `mod_python` request handler, you should redeploy your Django projects using another request handler. mod_wsgi is the request handler recommended by the Django project, but FastCGI is also supported. Support for `mod_python` deployment will be removed in Django 1.5.

Project Organization

or... where do apps belong?

Django Packages: A Case Study by Daniel Greenfeld. Since launch Django Packages has become the place to find and compare apps, frameworks and projects produced by the Django Community. Through the use of public APIs, Django Packages constantly fetches hard data from PyPI, Github, and Bitbucket, aproviding a powerful mash-up of real-world data on the volume of usage of a particular package. At a glance you can see which package is the most downloaded, which is the most used, and which has seen ongoing development. This talk is for everyone, including non-Django users, and covers everything from architecture, API development, to interacting with PyPI, Github, Bitbucket, etc. It will include tools, lessons learned, and projects that fork the code to save the world and will finish with an overview of the forthcoming pypackages.com.

Pluggable Django Patterns by Corey Oordt. **(Extreme)** Pluggable or reusable applications are a key feature of Django, but there is little guidance on writing them well. This talk will dig into the different types of Django applications and coding patterns that make writing a reusable application easier. This talk also covers ways to avoid common implementation gotchas.

```
$ django-admin.py startproject myproj
$ cd myproj/
$ ls
__init__.py manage.py   settings.py urls.py
$ django-admin.py startapp myapp
$ django-admin.py startapp anotherapp

$ ls
__init__.py anotherapp  manage.py   myapp
settings.py urls.py
```

```
$ tail settings.py
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    <...>
    'myapp',
    'anotherapp',
)
```

```
$ python manage.py shell
<...>
>>> pprint(sys.path)
['',
 '/Users/brian/work/temp/myproj',
 <...>
```

anywhere on sys.path works

up to you to decide how to organize things

recommendation:

as you're learning,
just keep everything
together

as soon as you want to
re-use something, break it
out into a separate app

Admin Customization

a few notable admin customizations

http://docs.djangoproject.com/en/1.2//ref/contrib/admin/

The `field_options` dictionary can have the following keys:

- **`fields`**

  A tuple of field names to display in this fieldset. This key is required.

  Example:

```
{
'fields': ('first_name', 'last_name', 'address', 'city', 'state'),
}
```

  To display multiple fields on the same line, wrap those fields in their own tuple. In this example, the `first_name` and `last_name` fields will display on the same line:

```
{
'fields': (('first_name', 'last_name'), 'address', 'city', 'state'),
}
```

  New in Django 1.2: Please, see the release notes

  `fields` can contain values defined in `readonly_fields` to be displayed as read-only.

http://docs.djangoproject.com/en/1.2//ref/contrib/admin/#django.contrib.admin.ModelAdmin.fieldsets

- A string representing an attribute on the model. This behaves almost the same as the callable, but `self` in this context is the model instance. Here's a full model example:

```
class Person(models.Model):
    name = models.CharField(max_length=50)
    birthday = models.DateField()

    def decade_born_in(self):
        return self.birthday.strftime('%Y')[:3] + "0's"
    decade_born_in.short_description = 'Birth decade'

class PersonAdmin(admin.ModelAdmin):
    list_display = ('name', 'decade_born_in')
```

http://docs.djangoproject.com/en/1.2//ref/contrib/admin/#django.contrib.admin.ModelAdmin.list_display

`ModelAdmin.list_editable`

New in Django 1.1: Please, see the release notes

Set `list_editable` to a list of field names on the model which will allow editing on the change list page. That is, fields listed in `list_editable` will be displayed as form widgets on the change list page, allowing users to edit and save multiple rows at once.

http://docs.djangoproject.com/en/1.2//ref/contrib/admin/#django.contrib.admin.ModelAdmin.list_editable

`ModelAdmin.readonly_fields`

New in Django 1.2: Please, see the release notes

By default the admin shows all fields as editable. Any fields in this option (which should be a `list` or `tuple`) will display its data as-is and non-editable. This option behaves nearly identical to `ModelAdmin.list_display`. Usage is the same, however, when you specify `ModelAdmin.fields` or `ModelAdmin.fieldsets` the read-only fields must be present to be shown (they are ignored otherwise).

If `readonly_fields` is used without defining explicit ordering through `ModelAdmin.fields` or `ModelAdmin.fieldsets` they will be added last after all editable fields.

http://docs.djangoproject.com/en/1.2//ref/contrib/admin/#django.contrib.admin.ModelAdmin.readonly_fields

`ModelAdmin.queryset`(*self, request*)

The `queryset` method on a `ModelAdmin` returns a `QuerySet` of all model instances that can be edited by the admin site. One use case for overriding this method is to show objects owned by the logged-in user:

```
class MyModelAdmin(admin.ModelAdmin):
    def queryset(self, request):
        qs = super(MyModelAdmin, self).queryset(request)
        if request.user.is_superuser:
            return qs
        return qs.filter(author=request.user)
```

http://docs.djangoproject.com/en/1.2//ref/contrib/admin/#django.contrib.admin.ModelAdmin.queryset

Overriding Admin Templates

It is relatively easy to override many of the templates which the admin module uses to generate the various pages of an admin site. You can even override a few of these templates for a specific app, or a specific model.

Set up your projects admin template directories

The admin template files are located in the `contrib/admin/templates/admin` directory.

In order to override one or more of them, first create an `admin` directory in your project's `templates` directory. This can be any of the directories you specified in `TEMPLATE_DIRS`.

Within this `admin` directory, create sub-directories named after your app. Within these app subdirectories create sub-directories named after your models. Note, that the admin app will lowercase the model name when looking for the directory, so make sure you name the directory in all lowercase if you are going to run your app on a case-sensitive filesystem.

To override an admin template for a specific app, copy and edit the template from the `django/contrib/admin/templates/admin` directory, and save it to one of the directories you just created.

http://docs.djangoproject.com/en/1.2//ref/contrib/admin/#overriding-admin-templates

Overriding vs. replacing an admin template

Because of the modular design of the admin templates, it is usually neither necessary nor advisable to replace an entire template. It is almost always better to override only the section of the template which you need to change.

To continue the example above, we want to add a new link next to the `History` tool for the `Page` model. After looking at `change_form.html` we determine that we only need to override the `object-tools` block. Therefore here is our new `change_form.html` :

```
{% extends "admin/change_form.html" %}
{% load i18n %}
{% block object-tools %}
{% if change %}{% if not is_popup %}
  <ul class="object-tools">
    <li><a href="history/" class="historylink">{% trans "History" %}</a></
    <li><a href="mylink/" class="historylink">My Link</a></li>
    {% if has_absolute_url %}
        <li><a href="../../../r/{{ content_type_id }}/{{ object_id }}/" cl
            {% trans "View on site" %}</a>
        </li>
    {% endif%}
  </ul>
{% endif %}{% endif %}
{% endblock %}
```

http://docs.djangoproject.com/en/1.2//ref/contrib/admin/#overriding-vs-replacing-an-admin-template

admin docs

http://docs.djangoproject.com/en/1.2//ref/contrib/admin/admindocs/

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    # Uncomment the next line to enable the admin:
    'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    'django.contrib.admindocs',
    'polls',
)
```

Documentation

localhost:8000/admin/doc/

Django administration

Welcome, **brian**. Documentation / Change password / Log out

Home › Documentation

Documentation

Tags

List of all the template tags and their functions.

Filters

Filters are actions which can be applied to variables in a template to alter the output.

Models

Models are descriptions of all the objects in the system and their associated fields. Each model has a list of fields which can be accessed as template variables.

Views

Each page on the public site is generated by a view. The view defines which template is used to generate the page and which objects are available to that template.

Bookmarklets

Tools for your browser to quickly access admin functionality.

Model: polls.Poll

localhost:8000/admin/doc/models/polls.poll/

Django administration

Welcome, **brian**. Documentation / Change password / Log out

Home › Documentation › Models › polls.Poll

Fields on Poll objects

Poll(id, question, pub_date)

Field	Type	Description
choice_set.all	List	all related polls.Choice objects
choice_set.count	Integer	number of related polls.Choice objects
id	Integer	ID
pub_date	Date (with time)	date published
question	String (up to 200)	question
was_published_today		None

‹ Back to Models Documentation

Template tags

localhost:8000/admin/doc/tags/

Django administration

Welcome, **brian**. Documentation / Change password / Log out

Home › Documentation › Tags

Template tag documentation

Built-in tags

- autoescape
- block
- comment
- csrf_token
- cycle
- debug
- extends
- filter
- firstof
- for
- if
- ifchanged
- ifequal
- ifnotequal
- include
- load
- now
- regroup
- spaceless
- ssi
- templatetag

Built-in tags

autoescape

Force autoescape behaviour for this block.

block

Define a block that can be overridden by child templates.

comment

Ignores everything between `{% comment %}` and `{% endcomment %}`.

csrf_token

cycle

Template tags

localhost:8000/admin/doc/tags/

- with

admin_list

- admin_actions
- admin_list_filter
- date_hierarchy
- pagination
- paginator_number
- result_list
- search_form

admin_modify

- prepopulated_fields_js
- submit_row

adminmedia

- admin_media_prefix

cache

- cache

i18n

- blocktrans
- get_available_languages
- get_current_language
- get_current_language_bidi
- trans

Within a loop, cycles among the given strings each time through the loop.

```
{% for o in some_list %}
    <tr class="{% cycle 'row1' 'row2' %}">
        ...
    </tr>
{% endfor %}
```

Outside of a loop, give the values a unique name the first time you call it, then use that name each sucessive time through:

```
<tr class="{% cycle 'row1' 'row2' 'row3' as rowcolors %}">...</tr>
<tr class="{% cycle rowcolors %}">...</tr>
<tr class="{% cycle rowcolors %}">...</tr>
```

You can use any number of values, separated by spaces. Commas can also be used to separate values; if a comma is used, the cycle values are interpreted as literal strings.

debug

Outputs a whole load of debugging information, including the current context and imported modules.

Sample usage:

```
<pre>
    {% debug %}
```

for django core, you can just look at their main docs, but most apps don't have that level of documentation, and this is a great reference

you can add docstrings with special annotations, and your code will be displayed just like the rest

http://docs.djangoproject.com/en/1.2//ref/contrib/admin/admindocs/#documentation-helpers

admin actions

http://docs.djangoproject.com/en/1.2//ref/contrib/admin/actions/

http://docs.djangoproject.com/en/1.2//ref/contrib/admin/actions/

Templates

starting to get
comfy with templates?

filters?

template inheritance?

custom template tags?

`{% url %}` ?

filters!

http://docs.djangoproject.com/en/1.2/topics/templates/#filters

http://docs.djangoproject.com/en/1.2/ref/templates/builtins/#built-in-filter-reference

```
{{ first_name|lower }}
```

converts the first_name variable to lower case and then displays it

```
{{ description|escape|linebreaks }}
```

escape the description, then convert line breaks to <br /> and <p> tags

there are a bunch of built-in filters

http://docs.djangoproject.com/en/1.2/ref/templates/builtins/#built-in-filter-reference

and you can write your own

http://docs.djangoproject.com/en/1.2/howto/custom-template-tags/
#writing-custom-template-filters

more examples

default

If value evaluates to `False`, use given default. Otherwise, use the value.

For example:

```
{{ value|default:"nothing" }}
```

If `value` is `""` (the empty string), the output will be `nothing`.

dictsort

Takes a list of dictionaries and returns that list sorted by the key given in the argument.

For example:

```
{{ value|dictsort:"name" }}
```

If `value` is:

```
[
    {'name': 'zed', 'age': 19},
    {'name': 'amy', 'age': 22},
    {'name': 'joe', 'age': 31},
]
```

then the output would be:

```
[
    {'name': 'amy', 'age': 22},
    {'name': 'joe', 'age': 31},
    {'name': 'zed', 'age': 19},
]
```

floatformat

When used without an argument, rounds a floating-point number to one decimal place -- but only if there's a decimal part to be displayed. For example:

value	Template	Output
34.23234	{{ value|floatformat }}	34.2
34.00000	{{ value|floatformat }}	34
34.26000	{{ value|floatformat }}	34.3

If used with a numeric integer argument, `floatformat` rounds a number to that many decimal places. For example:

value	Template	Output
34.23234	{{ value|floatformat:3 }}	34.232
34.00000	{{ value|floatformat:3 }}	34.000
34.26000	{{ value|floatformat:3 }}	34.260

join

Joins a list with a string, like Python's `str.join(list)`

For example:

```
{{ value|join:" // " }}
```

If `value` is the list `['a', 'b', 'c']`, the output will be the string `"a // b // c"`.

pluralize

Returns a plural suffix if the value is not 1. By default, this suffix is `'s'`.

Example:

```
You have {{ num_messages }} message{{ num_messages|pluralize }}.
```

If `num_messages` is `1`, the output will be `You have 1 message.` If `num_messages` is `2` the output will be `You have 2 messages.`

For words that require a suffix other than `'s'`, you can provide an alternate suffix as a parameter to the filter.

Example:

```
You have {{ num_walruses }} walrus{{ num_walruses|pluralize:"es" }}.
```

For words that don't pluralize by simple suffix, you can specify both a singular and plural suffix, separated by a comma.

Example:

```
You have {{ num_cherries }} cherr{{ num_cherries|pluralize:"y,ies" }}.
```

random

Returns a random item from the given list.

For example:

```
{{ value|random }}
```

If `value` is the list `['a', 'b', 'c', 'd']`, the output could be `"b"`.

slugify

Converts to lowercase, removes non-word characters (alphanumerics and underscores) and converts spaces to hyphens. Also strips leading and trailing whitespace.

For example:

```
{{ value|slugify }}
```

If `value` is `"Joel is a slug"`, the output will be `"joel-is-a-slug"`.

Other tags and filter libraries

Django comes with a couple of other template-tag libraries that you have to enable explicitly in your `INSTALLED_APPS` setting and enable in your template with the `{% load %}` tag.

django.contrib.humanize

A set of Django template filters useful for adding a "human touch" to data. See django.contrib.humanize.

django.contrib.markup

A collection of template filters that implement these common markup languages:

- Textile
- Markdown
- reST (reStructuredText)

See the markup documentation.

django.contrib.webdesign

A collection of template tags that can be useful while designing a Web site, such as a generator of Lorem Ipsum text. See django.contrib.webdesign.

i18n

Provides a couple of templatetags that allow specifying translatable text in Django templates. It is slightly different from the libraries described above because you don't need to add any application to the `INSTALLED_APPS` setting but rather set `USE_I18N` to True, then loading it with `{% load i18n %}`. See Specifying translation strings: In template code.

phone2numeric

Converts a phone number (possibly containing letters) to its numerical equivalent.

The input doesn't have to be a valid phone number. This will happily convert any string.

For example:

```
{{ value|phone2numeric }}
```

If `value` is `800-COLLECT`, the output will be `800-2655328`.

```
{{ first|add:second }}
```

We’re done. You can write my favorite example right in template tags! I thought it wasn’t supposed to be a real language! :)

inheritance

http://docs.djangoproject.com/en/1.2/topics/templates/#template-inheritance

Read about it and play with it in your own app. Just a taste here.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}My amazing site{% endblock %}</title>
</head>

<body>
    <div id="sidebar">
        {% block sidebar %}
        <ul>
            <li><a href="/">Home</a></li>
            <li><a href="/blog/">Blog</a></li>
        </ul>
        {% endblock %}
    </div>

    <div id="content">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

```
{% extends "base.html" %}

{% block title %}My amazing blog{% endblock %}

{% block content %}
{% for entry in blog_entries %}
    <h2>{{ entry.title }}</h2>
    <p>{{ entry.body }}</p>
{% endfor %}
{% endblock %}
```

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <link rel="stylesheet" href="style.css" />
    <title>My amazing blog</title>
</head>

<body>
    <div id="sidebar">
        <ul>
            <li><a href="/">Home</a></li>
            <li><a href="/blog/">Blog</a></li>
        </ul>
    </div>

    <div id="content">
        <h2>Entry one</h2>
        <p>This is my first entry.</p>

        <h2>Entry two</h2>
        <p>This is my second entry.</p>
    </div>
</body>
</html>
```

URLs

```
urlpatterns = patterns('',
    # Example:
    # (r'^mysite/', include('mysite.foo.urls')),
    (r'^polls/$', 'polls.views.index'),
    (r'^polls/(?P<poll_id>\d+)/$', 'polls.views.detail'),
    (r'^polls/(?P<poll_id>\d+)/results/$', 'polls.views.results'),
    (r'^polls/(?P<poll_id>\d+)/vote/$', 'polls.views.vote'),

    # Uncomment the admin/doc line below to enable admin doc:
    (r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    (r'^admin/', include(admin.site.urls)),
)
```

```
<html>
<body>
    <p><a href="/polls/">Index</a></p>
{% if latest_polls %}
    <ul>
    {% for poll in latest_polls %}
        <li><a href="/polls/{{ poll.id }}/">
{{ poll.question }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
</body>
</html>
```

“Recent polls indicate that our customers prefer the word ‘survey’ to ‘poll.’ Please change all the site URLS immediately.”

-- PHB

ouch!

```
<html>
<body>
    <p><a href="/polls/">Index</a></p>
{% if latest_polls %}
    <ul>
    {% for poll in latest_polls %}
        <li><a href="/polls/{{ poll.id }}/">
{{ poll.question }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
</body>
</html>
```

the URL template tag is your friend

http://docs.djangoproject.com/en/1.2//ref/templates/builtins/#url

```
urlpatterns = patterns('',
    # Example:
    # (r'^mysite/', include('mysite.foo.urls')),
    (r'^polls/$', 'polls.views.index'),
    (r'^polls/(?P<poll_id>\d+)/$', 'polls.views.detail'),
    (r'^polls/(?P<poll_id>\d+)/results/$', 'polls.views.results'),
    (r'^polls/(?P<poll_id>\d+)/vote/$', 'polls.views.vote'),

    # Uncomment the admin/doc line below to enable admin doc:
    (r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    (r'^admin/', include(admin.site.urls)),
)
```

```
<html>
<body>
    <p><a href="{% url polls.views.index %}">Index</a></p>
{% if latest_polls %}
    <ul>
    {% for poll in latest_polls %}
    <li><a href="{% url polls.views.detail poll.id %}">
{{ poll.question }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
</body>
</html>
```

and you can take it even further with names and namespaces, if you want

http://docs.djangoproject.com/en/1.2//topics/http/urls/#naming-url-patterns

```
from django.conf.urls.defaults import *

# Uncomment the next two lines to enable the admin:
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # Example:
    # (r'^mysite/', include('mysite.foo.urls')),
    url(r'^polls/$', 'polls.views.index', name='poll-index'),
    url(r'^polls/(?P<poll_id>\d+)/$', 'polls.views.detail',
                                      name='poll-detail'),
    (r'^polls/(?P<poll_id>\d+)/results/$', 'polls.views.results'),
    (r'^polls/(?P<poll_id>\d+)/vote/$', 'polls.views.vote'),

    # Uncomment the admin/doc line below to enable admin documentation:
    (r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    (r'^admin/', include(admin.site.urls)),
)
```

```
<html>
<body>
    <p><a href="{% url poll-index %}">Index</a></p>
{% if latest_polls %}
    <ul>
    {% for poll in latest_polls %}
    <li><a href="{% url poll-detail poll.id %}">
{{ poll.question }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
</body>
</html>
```

Query API

queries

http://docs.djangoproject.com/en/1.2/topics/db/queries/

you’ve done a bunch of them now

questions?

some time soon,
you should read through
this whole document:

http://docs.djangoproject.com/en/1.2/topics/db/queries/

Lab A

(20)

Lab A

Break

(10)

RANDOM

Google
uw internet programming in python
Search
About 149,000 results (0.34 seconds)
Advanced search
Everything
Images
Videos
News
Shopping
More
Seattle, WA
Change location
Show search tools
Internet Programming in Python
Jan 11, 2011 ... UW Certificate Program in Python Programming ... Python Standard Library - Internet Protocols and Support ...
briandorsey.info/uwpython/ - Cached
Certificate in Python Programming | Classroom, Downtown Seattle ...
Developed in Partnership with the UW Department of Computer Science ... Internet Programming in Python Classroom, Downtown Seattle, Winter 2011 ...
www.pce.uw.edu/prog.aspx?id=4189 - Cached
Internet Programming in Python | Courses & Short Programs ...
Professional and Continuing Education - University of Washington ...
www.pce.uw.edu/course.aspx?id=5027 - Cached
Show more results from uw.edu

Eliot
Northwest District
Pearl
NW Lovejoy St
NW Glisan St
Portland
W Burnside St
Goose Hollow
Portlandia
SW Broadway
SW Market St
Vista Ave
Marquam Nature Park
NE Weidler St
NE Grand Ave
NE Couch St
SE Grand Ave
SE 12th Ave
SE Morris
15th Ave
Interstate Ave
30
405
5
84
26
99E

Powell's City of Books
Chinatown
Saturday Market Administrative Offi
Burnside St
Voodoo Doughnut
SW Washington St
SW Alder St
SW Naito Pkwy
405
1D
Arlene Schnitzer Concert hall
Portlandia
2A
Portland Marriott Downtown Waterfront
South Park Blocks
SW Broadway
SW 6th Ave
SW 4th Ave
SW Clay St
SW Market St
RiverPlace
300
1C
26
5
99W

This is me. Thanks for keeping me going this quarter!

Lecture B
(20)

APPS

reusable apps rock

Django Packages : django re

djangopackages.com

Django Packages

Help | Log in or Sign Up

CMS
BLOGS
API CREATION
ADMIN INTERFACE
ANALYTICS
ASSET MANAGERS
AUTHENTICATION
AUTO-COMPLETE
AWARDS AND BADGES
BUILDOUT
CACHING
COMMENTING
DATABASE MIGRATION
DEPLOYMENT
DEVELOPER TOOLS
E-COMMERCE
EMAIL
ERROR HANDLING
FORMS
FORUMS
LAYOUT
MANAGERS
NEWSLETTER
NOSQL
PAYMENT PROCESSING
PHOTOS
PROFILES
PROJECT MANAGEMENT
SEARCH
SECURITY
SOCIAL
TAGGING
TEMPLATE ENGINE
ADAPTERS
TESTING TOOLS
THIRD-PARTY APIS
TREES AND GRAPHS
VERSIONING
WEBSERVER
WIDGETS
» MORE GRIDS...

Django Packages is a directory of reusable apps, sites, tools, and more for your Django projects.

648 packages and counting!

search

We need your help

Know of any packages not listed here? Add them now! It's quick and easy.

add package »

Or add a grid comparing the features of 2 or more similar packages.

add grid »

Package Categories

Apps (510)

Small components used to build projects. An app is anything that is installed by placing in settings.INSTALLED_APPS.

Frameworks (31)

Large efforts that combine many python modules or apps. Examples include Django, Pinax, and Satchmo. Most CMS falls into this category.

Other (76)

Other are not installed by settings.INSTALLED_APPS, are not frameworks or sites but still help Django in some way.

Projects (31)

This is for individual projects such as Django Packages, DjangoProject.com, and others.

latest 5

0 users — **django-pbx-manager**
A Django based PBX manager allowing you to manage Asterisk and FreeSWITCH systems.

2 users — **Impostor**
Django app that enables staff to log in as other users using their own credentials.

1 users — **tokenapi**
Add an API to your Django app using token-based authentication.

1 users — **django-geckoboard**
Geckoboard custom widgets for Django projects

0 users — **djftpd**
FTP-server using Django's users for authentication.

http://djangopackages.com/

where to start?

What's your favorite Django

jacobian.org/writing/favorite-apps/

What's your favorite Django app?

Jacob Kaplan-Moss

November 1, 2010

On Twitter, I asked, "what's your favorite third-party Django app?" Eight hours later, I've got about 50 replies.

I meant the question to be fairly open-ended — I deliberately didn't clarify what I meant by "favorite" — and I'm not *that* popular, so this is by no means an accurate sample of the Django community. Still, the answers are a bit interesting, so let's take a look:

The winners

The favorite was… (drumroll please)… South. By a landslide, actually: about a third of the folks who responded chose South. I'm not particularly surprised: schema migration is a problem most users face, and Django doesn't have anything in core to handle it. But that's another show.

http://jacobian.org/writing/favorite-apps/

that's all I'm going to say about them today

FORMS

working with forms

`<form>...</form>`

http **POST**

http://docs.djangoproject.com/en/1.2/topics/forms/

you can write them yourself
in templates

and inspect the http request
in your view

this is fundamental
web stuff

Django has extensive
form tools

- Display an HTML form with automatically generated form widgets.
- Check submitted data against a set of validation rules.
- Redisplay a form in the case of validation errors.
- Convert submitted form data to the relevant Python data types.

this stuff is tedious to write

Overview

The library deals with these concepts:

Widget

A class that corresponds to an HTML form widget, e.g. `<input type="text">` or `<textarea>`. This handles rendering of the widget as HTML.

Field

A class that is responsible for doing validation, e.g. an `EmailField` that makes sure its data is a valid e-mail address.

Form

A collection of fields that knows how to validate itself and display itself as HTML.

Form Media

The CSS and JavaScript resources that are required to render a form.

The library is decoupled from the other Django components, such as the database layer, views and templates. It relies only on Django settings, a couple of `django.utils` helper functions and Django's internationalization hooks (but you're not required to be using internationalization features to use this library).

http://docs.djangoproject.com/en/1.2/topics/forms/

```python
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField()
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

http://docs.djangoproject.com/en/1.2/topics/forms/

Using a form in a view

The standard pattern for processing a form in a view looks like this:

```
def contact(request):
    if request.method == 'POST': # If the form has been submitted...
        form = ContactForm(request.POST) # A form bound to the POST data
        if form.is_valid(): # All validation rules pass
            # Process the data in form.cleaned_data
            # ...
            return HttpResponseRedirect('/thanks/') # Redirect after POST
    else:
        form = ContactForm() # An unbound form

    return render_to_response('contact.html', {
        'form': form,
    })
```

http://docs.djangoproject.com/en/1.2/topics/forms/

There are three code paths here:
- If the form has not been submitted, an unbound instance of ContactForm is created and passed to the template.
- If the form has been submitted, a bound instance of the form is created usingrequest.POST. If the submitted data is valid, it is processed and the user is re-directed to a "thanks" page.
- If the form has been submitted but is invalid, the bound form instance is passed on to the template.

The distinction between bound and unbound forms is important. An unbound form does not have any data associated with it; when rendered to the user, it will be empty or will contain default values. A bound form does have submitted data, and hence can be used to tell if that data is valid. If an invalid bound form is rendered it can include inline error messages telling the user where they went wrong.
See Bound and unbound forms for further information on the differences between bound and unbound forms.

Extending the above example, here's how the form data could be processed:

```
if form.is_valid():
    subject = form.cleaned_data['subject']
    message = form.cleaned_data['message']
    sender = form.cleaned_data['sender']
    cc_myself = form.cleaned_data['cc_myself']

    recipients = ['info@example.com']
    if cc_myself:
        recipients.append(sender)

    from django.core.mail import send_mail
    send_mail(subject, message, sender, recipients)
    return HttpResponseRedirect('/thanks/') # Redirect after POST
```

http://docs.djangoproject.com/en/1.2/topics/forms/

Displaying a form using a template

Forms are designed to work with the Django template language. In the above example, we passed our `ContactForm` instance to the template using the context variable `form`. Here's a simple example template:

```
<form action="/contact/" method="post">{% csrf_token %}
{{ form.as_p }}
<input type="submit" value="Submit" />
</form>
```

The form only outputs its own fields; it is up to you to provide the surrounding `<form>` tags and the submit button.

```
<form action="/contact/" method="post">
<p><label for="id_subject">Subject:</label>
    <input id="id_subject" type="text" name="subject" maxlength="100" /></
<p><label for="id_message">Message:</label>
    <input type="text" name="message" id="id_message" /></p>
<p><label for="id_sender">Sender:</label>
    <input type="text" name="sender" id="id_sender" /></p>
<p><label for="id_cc_myself">Cc myself:</label>
    <input type="checkbox" name="cc_myself" id="id_cc_myself" /></p>
<input type="submit" value="Submit" />
</form>
```

http://docs.djangoproject.com/en/1.2/topics/forms/

customization

http://docs.djangoproject.com/en/1.2/topics/forms/

everyone has different things they want to do with their forms, so there are lots of ways to customize forms
field order, subset of fields, etc

forms from models

looks a lot like a model class, right?

```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField()
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

http://docs.djangoproject.com/en/1.2/topics/forms/

Django knows how to make form classes from model classes

```python
from django.db import models
from django.forms import ModelForm

TITLE_CHOICES = (
    ('MR', 'Mr.'),
    ('MRS', 'Mrs.'),
    ('MS', 'Ms.'),
)

class Author(models.Model):
    name = models.CharField(max_length=100)
    title = models.CharField(max_length=3, choices=TITLE_CHOICES)
    birth_date = models.DateField(blank=True, null=True)

    def __unicode__(self):
        return self.name

class Book(models.Model):
    name = models.CharField(max_length=100)
    authors = models.ManyToManyField(Author)

class AuthorForm(ModelForm):
    class Meta:
        model = Author

class BookForm(ModelForm):
    class Meta:
        model = Book
```

save()

save or update a model instance to the database

formsets

groups of forms

http://docs.djangoproject.com/en/1.2/topics/forms/formsets/

formset
--->

Change poll | Django site ad

localhost:8000/admin/polls/poll/2/

Django administration

Welcome, brian. Documentation / Change password / Log out

Home › Polls › Polls › What is the answer?

Change poll

History

Date published: Date: 2011-02-19 Today | Time: 11:37:17 Now |

Question: What is the answer?

Choices

Choice	Votes	Delete?
yes yes	0	☐
no no	0	☐
maybe maybe	0	☐
	0	
	0	
	0	

Add another Choice

Delete | Save and add another | Save and continue editing | Save

fancy forms

what if you want to customize your form fields?

http://docs.djangoproject.com/en/1.2/topics/forms/media/

this is how you do it

Form Media

Rendering an attractive and easy-to-use Web form requires more than just HTML - it also requires CSS stylesheets, and if you want to use fancy "Web2.0" widgets, you may also need to include some JavaScript on each page. The exact combination of CSS and JavaScript that is required for any given page will depend upon the widgets that are in use on that page.

This is where Django media definitions come in. Django allows you to associate different media files with the forms and widgets that require that media. For example, if you want to use a calendar to render DateFields, you can define a custom Calendar widget. This widget can then be associated with the CSS and JavaScript that is required to render the calendar. When the Calendar widget is used on a form, Django is able to identify the CSS and JavaScript files that are required, and provide the list of file names in a form suitable for easy inclusion on your Web page.

http://docs.djangoproject.com/en/1.2/topics/forms/media/

MIDDLEWARE

installed by default

```
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
)
```

A common HTTP request-response path. From last week's slides.

hooks to allow alternate paths through the request/response cycle

simplification

DEPLOYING

lies, damn lies and statistics

djangosites
Shiny Websites powered by Django

http://www.djangosites.org/stats/

http://www.djangosites.org/stats/

http://www.djangosites.org/stats/

http://www.djangosites.org/stats/

djangosites
Shiny Websites powered by Django

http://www.djangosites.org/stats/

http://www.djangosites.org/stats/

options

set it up yourself

tools from hosting provider

django hosting provider

mod_wsgi on a VM

WebFaction Control Panel

https://panel.webfaction.com/app_/create

> Account > Affiliate program > Usage > Domains / websites > E-mails > Databases > Support

> Home > Apps > Add

WebFaction Control Panel
Logged in as briandorsey [logout]

Name:*

(The app will be installed in $HOME/webapps/<name>)

App category:* Django

App type:* Django 1.2.5 (mod_wsgi 3.3/Python 2.7)

App doc:

This creates an Apache server with mod_wsgi 3.3 and Python 2.7 installed running Django 1.2.5.

The Django libraries are installed in the ~/webapps/<app_name>/lib/python2.7/django directory.

The front-end web server proxies incoming requests to the Apache server listening on an unprivileged port.

A cron job is created to restart the Apache server every 20 minutes if it is not already running.

To restart Apache, run: ~/webapps/<app_name>/apache2/bin/restart
To start Apache, run: ~/webapps/<app_name>/apache2/bin/start
To stop Apache, run: ~/webapps/<app_name>/apache2/bin/stop

Django documentation is available at: http://docs.webfaction.com/software/django/index.html
Python documentation is available at: http://docs.webfaction.com/software/python.html

Create Cancel

http://www.webfaction.com/

Google App Engine - Google

code.google.com/appengine/

Google App Engine

Home | Docs | FAQ | Articles | Blog | Community | Terms | Download

Run your web apps on Google's infrastructure.

Easy to build, easy to maintain, easy to scale.

Google App Engine enables you to build and host web apps on the same systems that power Google applications. App Engine offers fast development and deployment; simple administration, with no need to worry about hardware, patches or backups; and effortless scalability. Discover why developers are choosing App Engine.

Google App Engine for Business New!

Run your enterprise applications on Google's infrastructure.

- **Centralized administration:** A new, company-focused administration console lets you manage all the applications in your domain.
- **Reliability and support:** 99.9% uptime service level agreement, with premium developer support available.
- **Secure by default:** Only users from your Google Apps domain can access applications by default.
- **Pricing that makes sense:** Each application costs just $8 per user, per month up to a maximum of $1000 a month. Pay only for what you use.
- **Enterprise features:** Coming later this year - hosted SQL databases, SSL on your company's domain for secure communications, and access to advanced Google services.

Learn more about the preview of App Engine for Business.

Getting Started

1. Sign up for an App Engine account.
2. Download the App Engine SDK.
3. Read the Getting Started Guide.

Watch and Learn

Developing and deploying on Google App Engine.
Watch Now

links from AppEngine.reddit.com

- AppEngine vs. EC2 (an attempt to compare apples to oranges)
  8 points | 3 comments
- The Unofficial Guide to Migrating Off of Google App Engine
  12 points | 3 comments
- new map on www.appenginepeople.net with fusiontables
  6 points | comment
- Are Apps for the Google App Engine usable on other servers? - Stack Overflow
  17 points | comment
- why can't we see these "next gen" queries yet?why did google present them so long ago without any progress updates?
  12 points | 10 comments

Gondor — effortless product

gondor.io

GONDOR

effortless production Django hosting

Gondor is the production infrastructure that we at Eldarion have built for our own sites and those of our clients. We're making it available to other Django developers like you so you can focus on your site, not your infrastructure.

It supports:

- **revision control** via git or mercurial
- **dependency management** using pip
- **database migrations** via South or nashvegas
- **full backups** of your entire application
- asynchronous and scheduled **task execution**
- **full-text search** using Solr and django-haystack
- **caching** via memcached or redis

Follow @eldarion_team on twitter.

```
$ pip install gondor
$ gondor create example.com
$ gondor deploy example.com master
$ gondor status example.com
$ gondor sqldump example.com
$ gondor runcommand example.com shell
```

The Gondor Blog

Apply for Private Beta

ep.io: Smart Python hosting

www.ep.io

LOG IN PRICING FAQ DOCUMENTATION SUPPORT BLOG

ep.io
smart python hosting

Quick, painless Python deployment.

This is ep.io, the hosted WSGI hosting and deployment solution. We take your Django, Flask, Pylons or other WSGI code, run all the annoying hosting parts for you, and make sure everything's redundant and backed up.

In return, we bill you only for the processing time and bandwidth you actually use - we'll even give you detailed analytics of when and where those resources are being used. However, we're currently in a closed beta, so this is only available to those we invite.

If you'd like an invite, fill out the form below; we'll send you an invitation when we're ready for the next round of invites (usually a couple of weeks). In the meantime, feel free to have a look through our documentation and our plans for pricing.

Email:

Get on the invite list

team@ep.io · About · Team · Twitter · Copyright © Coding Ponies Ltd. 2010 - 2011

yes, I'm over-simplifying

a lot.

tons of good details here:

https://github.com/jacobian/django-deployment-workshop

TESTING

lots and lots of Python testing tools

lots of web testing tools
(some of them in Python)

not enough space to talk about them here

interest in a short talk next quarter?

Django specific testing

http://docs.djangoproject.com/en/1.2/topics/testing/

I'm just skimming the docs here, spending a bit of time with the testing docs will reward you greatly.

Writing tests

There are two primary ways to write tests with Django, corresponding to the two test frameworks that ship in the Python standard library. The two frameworks are:

- **Unit tests** – tests that are expressed as methods on a Python class that subclasses `unittest.TestCase`. For example:

```
import unittest

class MyFuncTestCase(unittest.TestCase):
    def testBasic(self):
        a = ['larry', 'curly', 'moe']
        self.assertEqual(my_func(a, 0), 'larry')
        self.assertEqual(my_func(a, 1), 'curly')
```

- **Doctests** -- tests that are embedded in your functions' docstrings and are written in a way that emulates a session of the Python interactive interpreter. For example:

```
def my_func(a_list, idx):
    """
    >>> a = ['larry', 'curly', 'moe']
    >>> my_func(a, 0)
    'larry'
    >>> my_func(a, 1)
    'curly'
    """
    return a_list[idx]
```

you'll probably end up mostly using unittest

Here is an example `unittest.TestCase` subclass:

```
import unittest
from myapp.models import Animal

class AnimalTestCase(unittest.TestCase):
    def setUp(self):
        self.lion = Animal.objects.create(name="lion", sound="roar")
        self.cat = Animal.objects.create(name="cat", sound="meow")

    def testSpeaking(self):
        self.assertEqual(self.lion.speak(), 'The lion says "roar"')
        self.assertEqual(self.cat.speak(), 'The cat says "meow"')
```

Running tests

Once you've written tests, run them using the `test` command of your project's `manage.py` utility:

```
$ ./manage.py test
```

By default, this will run every test in every application in `INSTALLED_APPS`. If you only want to run tests for a particular application, add the application name to the command line. For example, if your `INSTALLED_APPS` contains `'myproject.polls'` and `'myproject.animals'`, you can run the `myproject.animals` unit tests alone with this command:

```
$ ./manage.py test animals
```

Note that we used `animals`, not `myproject.animals`.

You can be even *more* specific by naming an individual test case. To run a single test case in an application (for example, the `AnimalTestCase` described in the "Writing unit tests" section), add the name of the test case to the label on the command line:

Testing tools

Django provides a small set of tools that come in handy when writing tests.

The test client

The test client is a Python class that acts as a dummy Web browser, allowing you to test your views and interact with your Django-powered application programmatically.

Some of the things you can do with the test client are:

- Simulate GET and POST requests on a URL and observe the response -- everything from low-level HTTP (result headers and status codes) to page content.
- Test that the correct view is executed for a given URL.
- Test that a given request is rendered by a given Django template, with a template context that contains certain values.

Note that the test client is not intended to be a replacement for Twill, Selenium, or other "in-browser" frameworks. Django's test client has a different focus. In short:

- Use Django's test client to establish that the correct view is being called and that the view is collecting the correct context data.
- Use in-browser frameworks such as Twill and Selenium to test *rendered* HTML and the *behavior* of Web pages, namely JavaScript functionality.

A comprehensive test suite should use a combination of both test types.

recommendation:

don't try to build a comprehensive suite in doc strings - use unittest or stand alone text files with doctest sessions in them

It takes away from the documentation to have every single corner case test included in the docs. Essays with tests in them, however, are awesome.

when to test?

how?

how much is enough?

it depends

sorry

here's a place to start:

“I don't do test-driven development;

I do stupidity-driven testing.

When I do something stupid, I write a test to make sure I don't do it again.”

-- Titus Brown
PyCon 2007 - testing panel.

http://ivory.idyll.org/blog/feb-07/stupidity-driven-testing.html

“For readers that don't get it, my development practice is this:
- write code to solve some problem
- watch code break in some obvious way
- write a test that tests that specific breakage
- lather, rinse, repeat.
I don't mind making mistakes, even stupid ones: I just don't want to repeat them. Thus, this development technique.”

more good stuff here:

http://docs.djangoproject.com/en/1.2/topics/testing/

DEBUGGING

Go read this:

http://simonwillison.net/2008/May/22/debugging/

command line debugger

```
# only useful when running the dev server locally
import pdb; pdb.set_trace()
```

http://onlamp.com/pub/a/python/2005/09/01/debugger.html

GUI - IDE

Debug - /home/luke/devel/clients/cardiff_university/extsrcs/hg/django-trunk/django/db/models/base.py - Eclipse SDK

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

Debug

MainThread - pid16208_seq2
- save_base [base.py:521]
- save_base [models.py:59]
- save_base [base.py:476]
- save_base [models.py:59]
- save [base.py:432]
- save [pluginmodel.py:137]
- save_instance [models.py:95]
- save [models.py:288]
- save_form [options.py:587]

Variables | Breakpoints

Name	Value
force_update	bool: False
manager	Manager: <django.db.models.manager.Manager object at 0x949bc4c>
meta	Options: cms.cmsplugin
non_pks	list: [<django.db.models.fields.BooleanField object at 0x949b78c>, <django.db.mc

Manager: <django.db.models.manager.Manager object at 0x949bc4c>

models.py | base.py | base.py | util.py | settings.py

```
            values = [(f, f.get_db_prep_save(raw and getattr(self, f.attname) or f.pre_save(self, True), connection=connection))
                for f in meta.local_fields]

        if meta.order_with_respect_to:
            field = meta.order_with_respect_to
            values.append((meta.get_field_by_name('_order')[0], manager.using(using).filter(**{field.name: getattr(self, field.attname)}
        record_exists = False

        update_pk = bool(meta.has_auto_field and not pk_set)
        if values:
            # Create a new record.
            result = manager._insert(values, return_id=update_pk, using=using)
        else:
            # Create a new record with
            result = manager._insert([(                                                    sing=using)
```

list (base)

list (base)

list (base)

list: [(<django.db.models.fields.BooleanField object at 0x949b78c>, True), (<django.db.models.fields.related.OneToOneField object at 0x949b8ac>, None), (<django.db.models.fields.SmallIntegerField object at 0x949ba0c>, 1), (<django.db.models.fields.related.ForeignKey object at 0x94947ec>, None), (<django.db.models.fields.related.ForeignKey object at 0x949484c>, None), (<django.db.models.fields.PositiveSmallIntegerField object at 0x949492c>, None), (<django.db.models.fields.CharField object at 0x94949cc>, ''), (<django.

Outline
- types
- sys
- os
- izip (itertools)
- django.db.models.manager
- ObjectDoesNotExist,MultipleObjectsReturned,Fie
- validators (django.core)
- AutoField,FieldDoesNotExist (django.db.models.
- OneToOneRel,ManyToOneRel,OneToOneField (dja
- delete_objects,Q (django.db.models.query)

Console | Tasks

manage.py

```
[08/Jan/2010 21:45:36] "GET /admin/jsi18n/ HTTP/1.1
[08/Jan/2010 21:45:36] "GET /site_media_files/cms/j
[08/Jan/2010 21:46:44] "POST /admin/cms/page/1/edit
[08/Jan/2010 21:52:09] "POST /admin/cms/page/1/edit
[08/Jan/2010 21:52:09] "GET /admin/cms/page/1/edit-plugin/8/edit-plugin/24/?_popup=1 HTTP/1.1" 200 4834
[08/Jan/2010 21:52:10] "GET /admin/jsi18n/ HTTP/1.1" 200 1861
[08/Jan/2010 21:52:10] "GET /site_media_files/cms/js/lib/jquery.js HTTP/1.1" 304 0
```

http://lukeplant.me.uk/blog/posts/eclipse-pydev-and-virtualenv/

GUI - stand alone

simple.py (~) - Winpdb 1.4.6 - Tychod

File Breakpoints Control Window Help

Filter: Medium | Encoding: auto | Synchronicity: True

Namespace

Locals | Globals | Exception

Name	Type	Repr
+ __builtins__	module	<module '__builtin__' (built-in)>
__doc__	NoneType	None
__file__	str	b'/home/chris/simple.pyc'
__name__	str	b'__main__'
__package__	NoneType	None

Source /home/chris/simple.py

```
1   #!/usr/bin/env python
2
3 c """A simple script."""
4
5   a = 1
6   b = 2
7
8   c = a + b
9
10  print c
11
12
```

Threads

TID	Name	State
140056081524480	MainThread	waiting at break point

Stack

Frame	Filename	Line	Function	Path
0	simple.py	3	<module>	/home/chris
1	rpdb2.py	14235	StartServer	/usr/share/pyshared
2	rpdb2.py	14485	main	/usr/share/pyshared
3	rpdb2.py	14514	<module>	/usr/lib/pymodules/pytho...

Console

```
RPDB2 - The Remote Python Debugger, ver
Copyright (C) 2005-2009 Nir Aides.
Type "help", "copyright", "license", "c

*** NEW: Use CTRL-N for auto completion
*** eval and exec.
*** Password has been set to a random p
*** Starting debuggee...
*** Attaching to debuggee...
*** Debug Channel is NOT encrypted.
*** Debuggee is waiting at break point
*** Successfully attached to '/home/chr
```

Command:

State: WAITING AT BREAK PC

http://code.google.com/p/winpdb/wiki/DebuggingTutorial

django-debug-toolbar by robhudson

A configurable set of panels that display various debug information about the current request/response.

Screencast by Idan Gazit

Lab B

(20)

2:35

```
INTERNAL_IPS = ('127.0.0.1',)

INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    # Uncomment the next line to enable the admin:
    'django.contrib.admin',
    # Uncomment the next line to enable admin docs:
    'django.contrib.admindocs',
    'polls',
    'debug_toolbar',
)
```

```
MIDDLEWARE_CLASSES = (
    'debug_toolbar.middleware.DebugToolbarMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    #'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
)
```

Cross Site Request Forgeries - http://docs.djangoproject.com/en/1.2/ref/contrib/csrf/

Lab B

- Install the Django Debug Toolbar: `http://robhudson.github.com/django-debug-toolbar/`
- Configure it
- Browse around your site and look at various panels, check the SQL
- Is Django doing what you'd expect? Did you learn something new?

WRAPUP

2:55

Info

Office hours:
Sunday 2-5pm

Top Pot Donuts
2124 5th Ave
Seattle, WA 98121
206-728-1966

turn in here: `http://is.gd/uwipip_week8`

Assignment

- port your week 7 or week 8 application to a different database backend

  next:

- A: Continue hacking on your week 8 application, get it setup on semi-permanent hosting, add polish, etc.

- B: Install redis and redis-py.
  Experiment with the list, set, sorted set and hash data types. How are they like Python datatypes? JSON? How are they different?
  Make two scripts which cooperate via the DB.
  (see week 9 extra reading for ideas)
  (the server is linux/mac only - use a VM if needed)